Linaro connect
Vancouver 2018

YVR18-417: Struck entropy!
Finding true randomness from sensor data

Sumit Garg and Daniel Thompson

Agenda

- Platform: Developerbox
- OP-TEE port
- OP-TEE use-case: RNG feasible?
- RNG mechanism
- Is this RNG truly random?
- Optimize RNG collection
- RNG use-cases

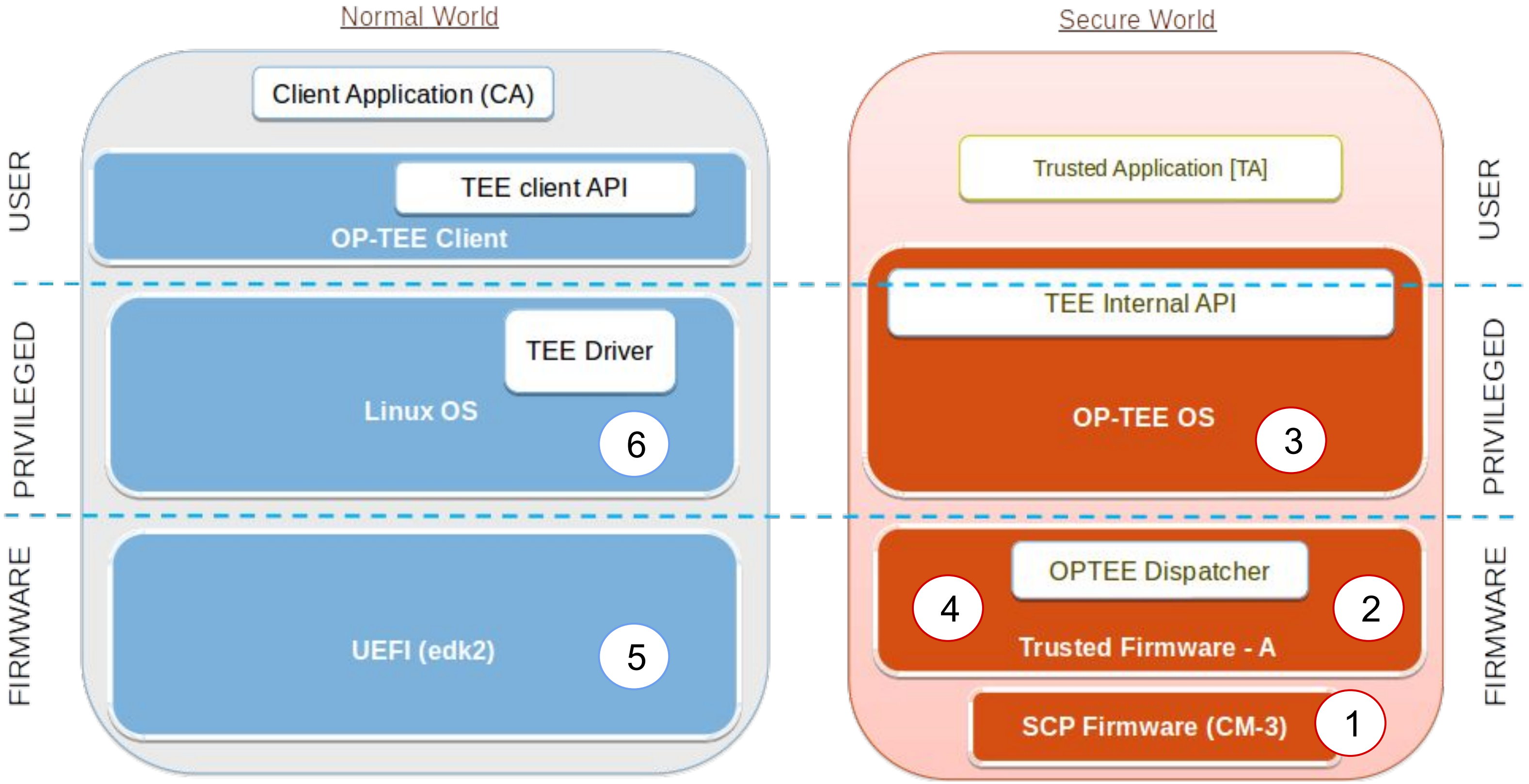
Platform: Developerbox

Based on Socionext SynQuacer SC2A11 multi-core chip with 24 cores of ARM® Cortex-A53.

Use-cases:

- ARM ® based software development environment.
- IoT gateway
- Edge computing
- Low power consumption server.

OP-TEE port for Developerbox



Note: Here numbering represents the boot sequence

OP-TEE use-case: RNG?

RNG – Random Number Generator

Developerbox lacks a hardware based TRNG.

Kernel provides a software implementation using randomness from inter-interrupt timings with following shortcomings:

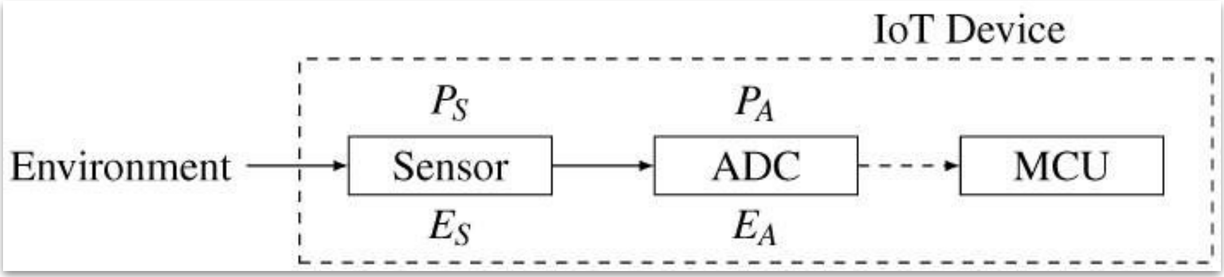
- Lacks sufficient entropy at critical points (especially at boot)
- Not trusted (eg. by OP-TEE)
- Quite slow (especially when there are few interrupts)

OP-TEE use-case: RNG feasible

Developerbox provides 7 on-chip thermal sensors, accessible from secure world only, sensing temperature from various group of core clusters.

Do these thermal sensors contain sufficient noise to develop a TRNG?

Sensor: Randomness sources

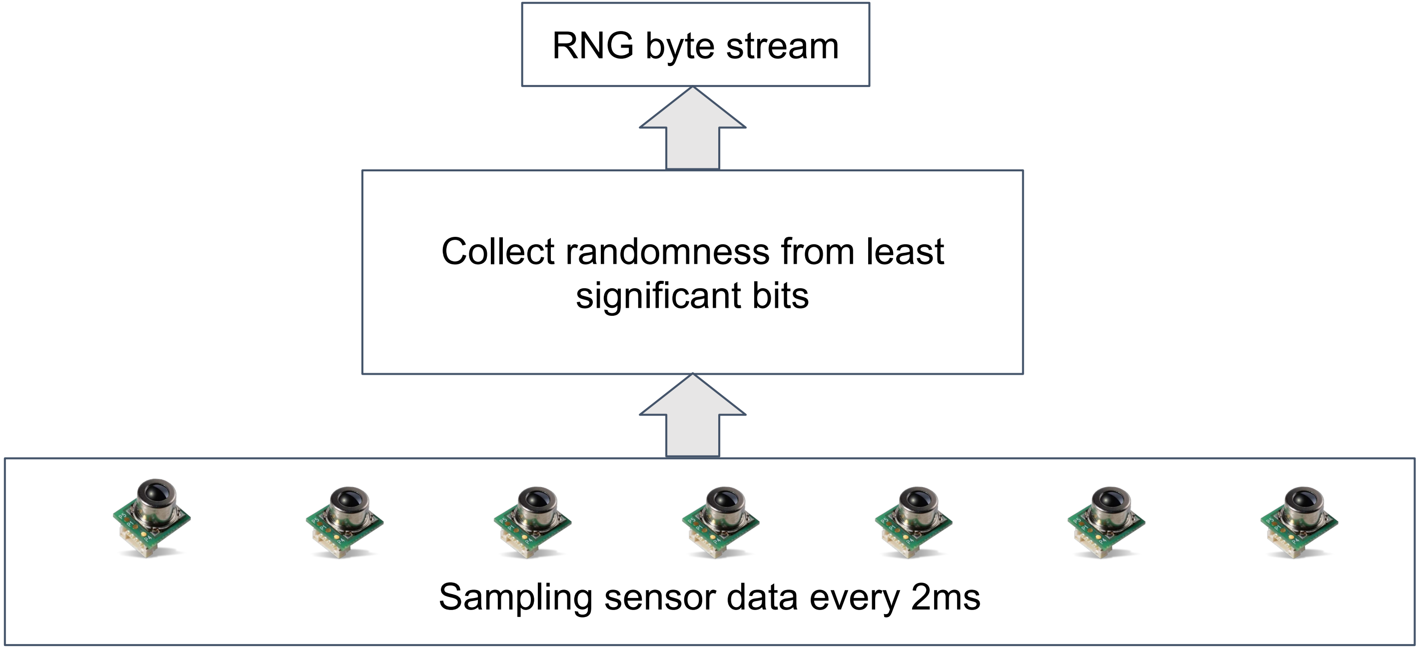


Randomness (measurement error + ADC conversion error) resides in Least Significant Bits (LSBs) of sensor output depending on precision of measurement and ADC conversion.

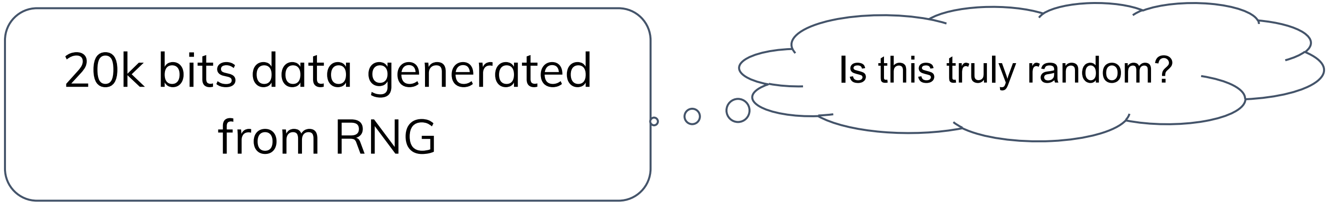
Thermal sensor: raw data

Sample no. (freq: 2ms)	Raw binary data							
S1	1	0	1	0	0	0	0	0
S2	1	0	0	1	1	1	1	1
S3	1	0	1	0	0	0	1	0
S4	1	0	1	0	0	0	1	1
S5	1	0	1	0	0	0	0	1
S6	1	0	1	0	0	0	1	0
S7	1	0	1	0	0	0	0	1
S8	1	0	1	0	0	0	1	1
S9	1	0	1	0	0	0	1	0
S10	1	0	1	0	0	0	1	1

RNG mechanism



Is this RNG truly random?



Answer to this really depends on how much paranoia one has. In our case we used following procedure to measure randomness:

- Collected approx. 2.1GB raw data from thermal sensors.
- Used “rngtest” (implements FIPS 140-2 RNG fitness tests).
  - https://wiki.archlinux.org/index.php/Rng-tools
  - https://en.wikipedia.org/wiki/FIPS_140-2

RNG algo 1: LSB only

FIPS test (rngtest) results show 2.13% success ratio.

Byte 0 (1st & 2nd sample)

G0	F0	E0	D0	C0	B0	A0	G0

Byte 1 (2nd & 3rd sample)

F0	E0	D0	C0	B0	A0	G0	F0

So on ...

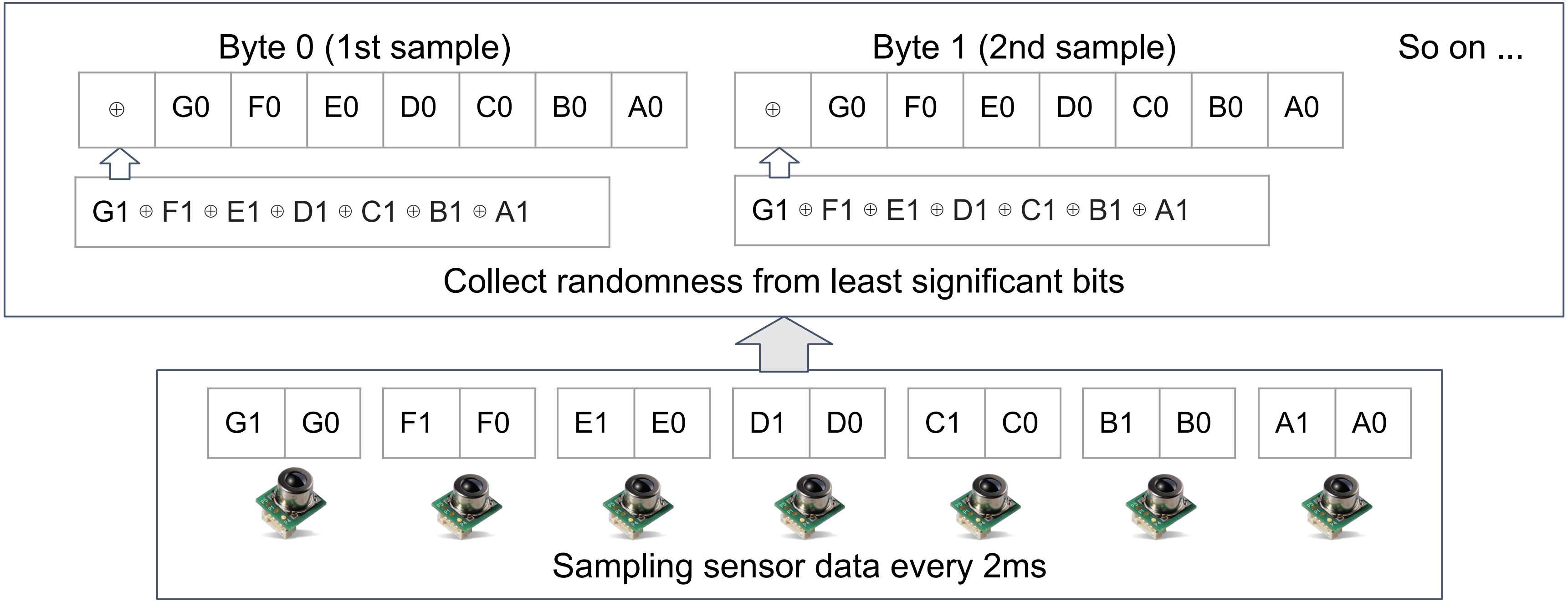
Collect randomness from least significant bits

G0	F0	E0	D0	C0	B0	A0

Sampling sensor data every 2ms

RNG algo 2: LSB + xor of bit1

FIPS test (rngtest) results show 0.07% success ratio.



RNG algo 3: LSB + xor of bit1 + CRC

Statistical fine tuning using CRC32 algo per 4 byte payload (32:32 bit whitening).

Statistical fine tuning using CRC32 algo per 5 byte payload (40:32 bit whitening).

Byte 0 (1st sample)

⊕	G0	F0	E0	D0	C0	B0	A0

G1 ⊕ F1 ⊕ E1 ⊕ D1 ⊕ C1 ⊕ B1 ⊕ A1

Byte 1 (2nd sample)

⊕	G0	F0	E0	D0	C0	B0	A0

G1 ⊕ F1 ⊕ E1 ⊕ D1 ⊕ C1 ⊕ B1 ⊕ A1

So on ...

Collect randomness from least significant bits

G1	G0	F1	F0	E1	E0	D1	D0	C1	C0	B1	B0	A1	A0

Sampling sensor data every 2ms

RNG testing results

- FIPS test (rngtest) results show **99.91%** success ratio.
- Entropy collection rate: **~500 bytes/sec**
- Tried using the dieharder suite to discriminate between the 32:32 bit and the 40:32 bit CRC whitening but results are ambiguous.

```
sumit@oak:~/latest$
sumit@oak:~/latest$
sumit@oak:~/latest$ rngtest -c 1000000 < entropy.2G_crc
rngtest 2-unofficial-mt.14
Copyright (c) 2004 by Henrique de Moraes Holschuh
This is free software; see the source for copying conditions.  There is NO warranty;

rngtest: starting FIPS tests...
rngtest: entropy source exhausted!
rngtest: bits received from input: 2161098752
rngtest: FIPS 140-2 successes: 107959
rngtest: FIPS 140-2 failures: 95
rngtest: FIPS 140-2(2001-10-10) Monobit: 12
rngtest: FIPS 140-2(2001-10-10) Poker: 9
rngtest: FIPS 140-2(2001-10-10) Runs: 41
rngtest: FIPS 140-2(2001-10-10) Long run: 33
rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
rngtest: input channel speed: (min=353.213; avg=3593.182; max=6357.829)Mibits/s
rngtest: FIPS tests speed: (min=10.930; avg=11.481; max=11.602)Mibits/s
rngtest: Program run time: 180154442 microseconds
sumit@oak:~/latest$
sumit@oak:~/latest$
sumit@oak:~/latest$
sumit@oak:~/latest$
```

Comparison with random.org data

- FIPS test (rngtest) results show similar **99.92%** success ratio.

```
rngtest 6
Copyright (c) 2004 by Henrique de Moraes Holschuh
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.

rngtest: starting FIPS tests...
rngtest: entropy source drained
rngtest: bits received from input: 20166213632
rngtest: FIPS 140-2 successes: 1007529
rngtest: FIPS 140-2 failures: 781
rngtest: FIPS 140-2(2001-10-10) Monobit: 103
rngtest: FIPS 140-2(2001-10-10) Poker: 102
rngtest: FIPS 140-2(2001-10-10) Runs: 292
rngtest: FIPS 140-2(2001-10-10) Long run: 290
rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
rngtest: input channel speed: (min=36.893; avg=18053.932;
max=19073.486)Mibits/s
rngtest: FIPS tests speed: (min=84.396; avg=161.269; max=178.257)Mibits/s
rngtest: Program run time: 120430395 microseconds
```

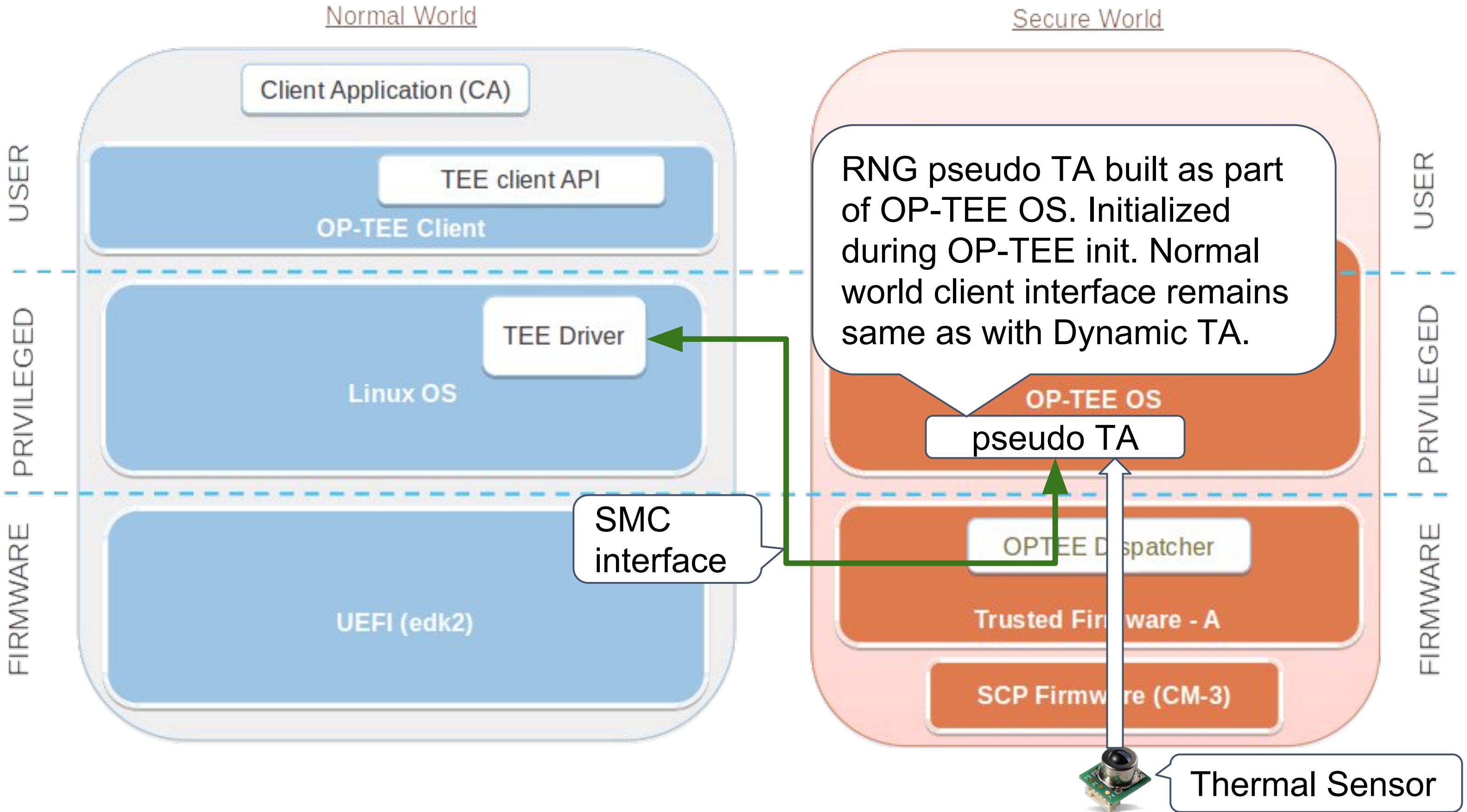
Aside: My Intel machine's RNG

- Uses "rdrand" hardware instruction to get entropy.
- FIPS test (rngtest) results show similar **99.91%** success ratio.
- Entropy collection rate far higher: **~1 Mbytes/sec**

```
sumit@sumit-XPS:~$ sudo rngtest -c 1000000 < /dev/random
rngtest 5
Copyright (c) 2004 by Henrique de Moraes Holschuh
This is free software; see the source for copying conditions.  There is NO warranty;

rngtest: starting FIPS tests...
^Crngtest: bits received from input: 6725620032
rngtest: FIPS 140-2 successes: 335997
rngtest: FIPS 140-2 failures: 284
rngtest: FIPS 140-2(2001-10-10) Monobit: 40
rngtest: FIPS 140-2(2001-10-10) Poker: 28
rngtest: FIPS 140-2(2001-10-10) Runs: 109
rngtest: FIPS 140-2(2001-10-10) Long run: 109
rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
rngtest: input channel speed: (min=3.630; avg=8.238; max=10.039)Mibits/s
rngtest: FIPS tests speed: (min=35.585; avg=94.055; max=101.997)Mibits/s
rngtest: Program run time: 846848802 microseconds
sumit@sumit-XPS:~$
sumit@sumit-XPS:~$
```

RNG implementation



Issue with RNG collection

- Sensor values refresh every 2ms
    - Theoretic maximum RNG rate of ~500 bytes/sec is relatively low
    - Max rate is only achieved if we read (poll) the sensors frequently

- Earlier implementation relied on continuous busy looping in pseudo TA until requested RNG data is generated (time: 2ms * no. of bytes)
    - Busy looping for multi-milliseconds is wasteful

Busy looping? Yuck!

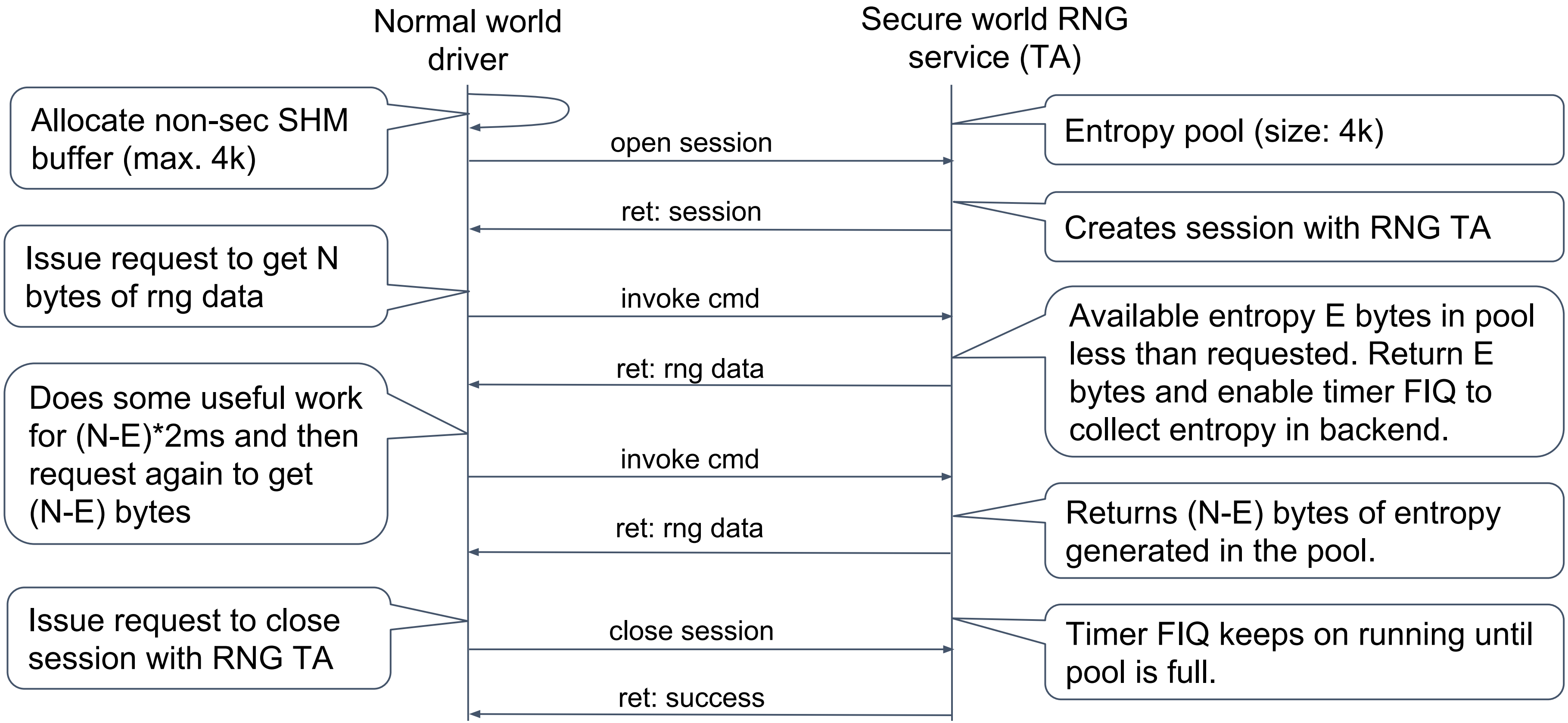
Optimize RNG collection

Configured OP-TEE pseudo TA with secure timer interrupts (freq: every 2ms) and entropy pool (size: 4k).

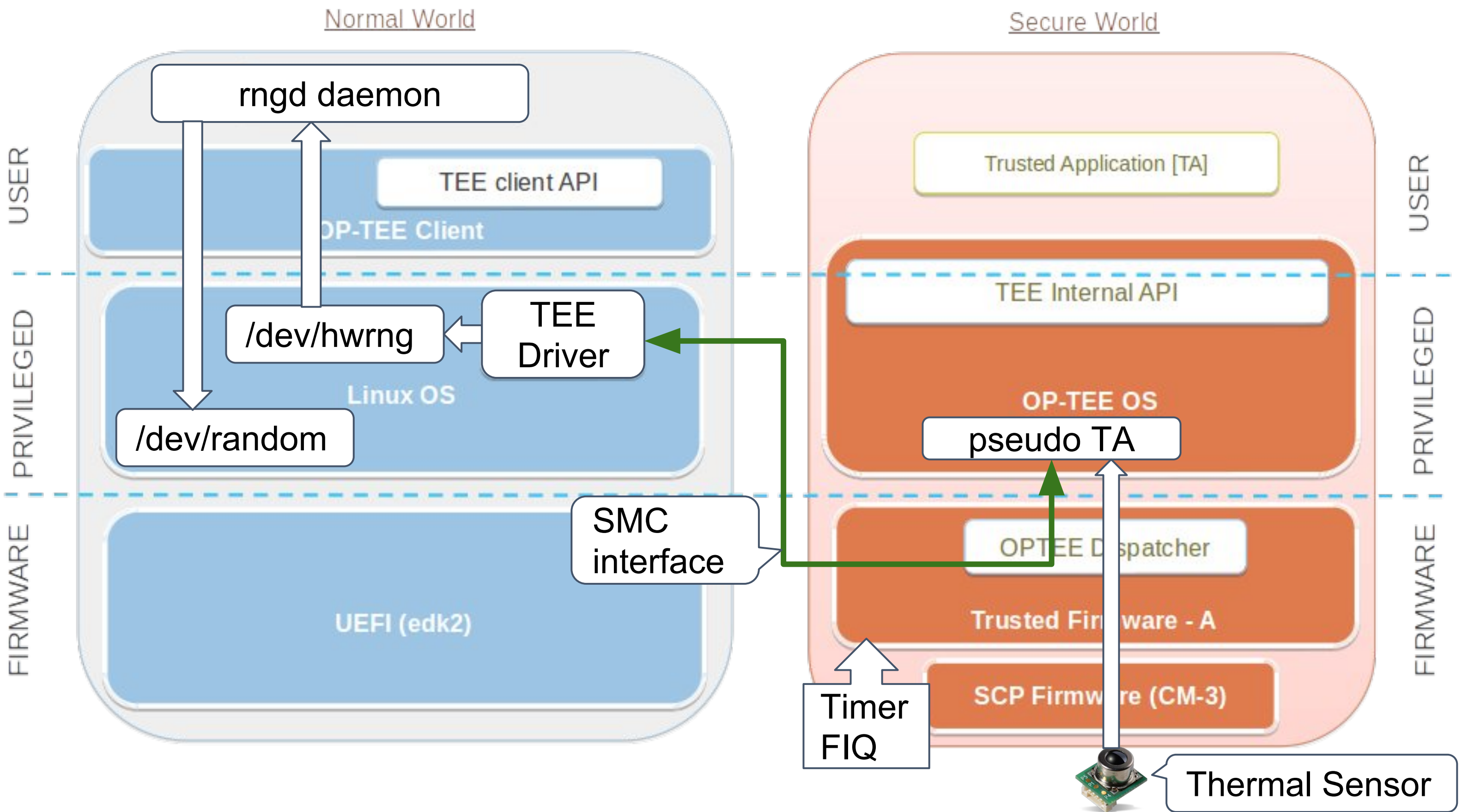
Functionality:

- At each interrupt, pseudo TA formulates byte from LSBs of thermal sensor output.
- Entropy pool is used to collect these bytes. Once pool is full, interrupts are disabled and enabled again with every entropy request.

RNG: SMC interface



RNG use-case - Kernel: /dev/random

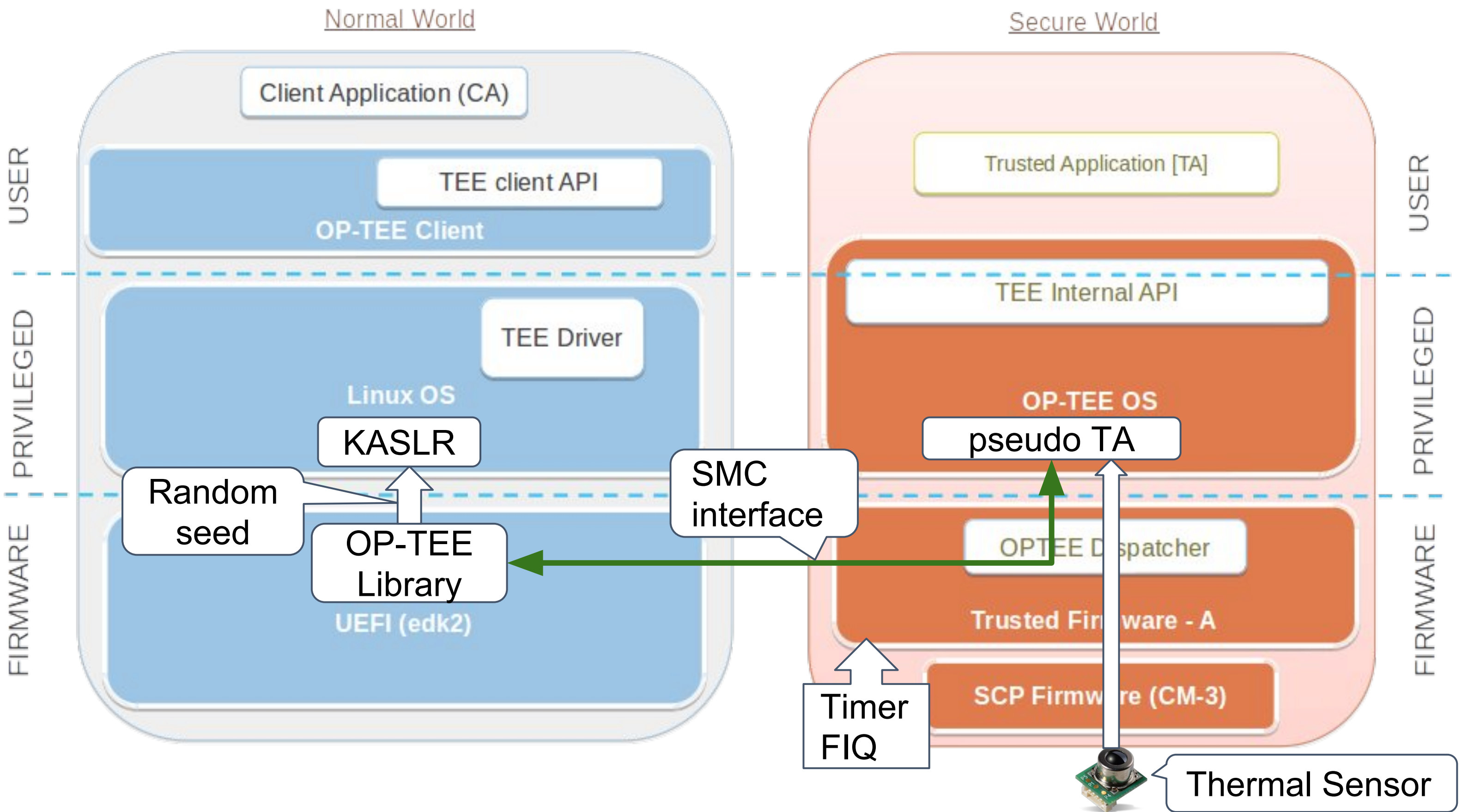


“rngtest” results for /dev/random

```
sumit@oak:~/latest$
sumit@oak:~/latest$ sudo rngtest -c 1000 < /dev/random
rngtest 2-unofficial-mt.14
Copyright (c) 2004 by Henrique de Moraes Holschuh
This is free software; see the source for copying conditions.  There is NO warranty;

rngtest: starting FIPS tests...
rngtest: bits received from input: 20000032
rngtest: FIPS 140-2 successes: 1000
rngtest: FIPS 140-2 failures: 0
rngtest: FIPS 140-2(2001-10-10) Monobit: 0
rngtest: FIPS 140-2(2001-10-10) Poker: 0
rngtest: FIPS 140-2(2001-10-10) Runs: 0
rngtest: FIPS 140-2(2001-10-10) Long run: 0
rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
rngtest: input channel speed: (min=1.535; avg=2.338; max=5709.222)Kibits/s
rngtest: FIPS tests speed: (min=11.207; avg=11.391; max=11.560)Mibits/s
rngtest: Program run time: 8356993481 microseconds
sumit@oak:~/latest$
```

RNG use-case - Boot time: UEFI



Next steps...

- Improvements to the whitening algorithm.
- Work towards creating a generic RNG interface to secure world handling:
  - Fast vs. slow entropy sources.
  - Implements entropy pool vs. must be called every N ms.
- Upstream RNG driver in edk2 (UEFI) and Linux.

Linaro connect
Vancouver 2018

Thank You

#YVR18
YVR18 keynotes and videos on: connect.linaro.org
For further information: www.linaro.org
Contact: support@linaro.org